Co:Z® Co-Processing Toolkit for z/OS

IBM Ported Tools OpenSSH - Quick Install Guide

1.0.4 Edition

Published February 25, 2015

Table of Contents

Revision History

Version 1.0.4 - February 25, 2015

- Added z/OS Ported Tools - OpenSSH specific version notification.

Version 1.0.3 - Sept 15, 2014

- Corrected some minor typos.

Version 1.0.2 - Aug 1, 2014

- Added notes on `CRYPTOZ/CLEARKEY.SYSTOK-SESSION-ONLY` dependency in ICSF HCR77A0+
- Added note on ICSF HCR77A1 option to disable SAF checking of RNG and MAC generation

Version 1.0.1 - Jun 9, 2014

- Corrected CPACF feature number
- Corrected the TCP configuration section to use the `NOAUTOLOG` keyword rather than `NOAUTO`

Version 1.0.0 - March 18, 2014

- Initial release

1. Basic Installation and Configuration

1.1 Introduction

This guide is designed to help systems programmers quickly install and configure *"IBM Ported Tools for z/OS - OpenSSH"*. Steps are taken to ensure smooth operation and efficiency, including exploitation of `ICSF` and `CPACF`. Proper configuration of these features has been shown to reduce CPU consumption for Ported Tools SFTP or Co:Z SFTP by over 50%.

While the procedures in this document will work in most environments, users should reference the appropriate IBM documentation as appropriate. The primary reference is the *IBM Ported Tools for z/OS: OpenSSH User's Guide*, ("PTUG"). This guide will call out specific sections of the PTUG or other documents for additional information.

Specific to z/OS Ported Tools - OpenSSH v1.2

This version of the quick install guide is specific to z/OS Ported Tools - OpenSSH v1.2, which became generally available in 2008. Please refer to version 1.3.x of this guide (available on our website) if you are using z/OS Ported Tools - OpenSSH v1.3

Topics covered in this guide:

- Prerequisites, service planning
- Language Environment tuning considerations
- ICSF support for secure random numbers via `/dev/random`
- Configuration files, started task, etc.
- z/OS Communications Server TCP/IP, Resolver and syslogd considerations
- ICSF support for hardware accelerated ciphers and MACs
- Managing the `/tmp` filesystem

Note: The included examples assume that you are running RACF as your system security product. IBM Ported Tools OpenSSH will also work with *CA-ACF2* and *CA-TSS*, but you will be required to translate RACF commands as shown to those products. If you have one of those products and would like to contribute tested examples, please contact us.

1.2 Prerequisites

This guide assumes that you have or will be installing IBM Ported Tools OpenSSH release 1.2, with PTF UA63842 (ICSF Crypto enhancements). Using this product and exploiting these features requires:

- z/OS 1.10 or later
- CPACF - processor feature 3863 (free and enabled by default in most countries)
- ICSF installed and running (even if you don't have a co-processor card)
  - CPACF instructions are used by ICSF for Ciphers and MACS
  - HCR77A0 ("A0" level) and later has support for `/dev/random` without crypto card. Requires z/OS 1.12 or

later.

1.3 Install / Service Planning

- Order/Install latest release: "IBM Ported Tools for z/OS" 5665-M23 1.2.0
  - Only the base FMID (HOS1120) is required.
  - This is a no-charge product and is included with a CBPDO.
  - The Program Directory: *www-03.ibm.com/systems/resources/fotza206.pdf*
  - See Upgrade: PORTED4ZOS Subset: HOS1120
- Be sure to include PTF UA63842 (FMID HOS1120) -
  - OA37278 - ICSF support for Ciphers and MACs
  - OA36257 - SMF recording in nested session
  - OA34819 - DOC; heap management issues
- If running on z/OS 1.10 or z/OS 1.11, check that the PTFs for APARs PK86329 and OA29401 have been applied
- Review and install as appropriate ICSF and its required service.

1.4 Check file attributes and ownership

From a z/OS Unix shell, check the permissions and owner of the following directories:

```
$ ls -ld /etc/ssh /var/empty /var/run
drwxr-xr-x❶  2 DB2ADM❷  STCGROUP    8192 Jul  3  2013 /etc/ssh
drwxr-xr-x   3 DB2ADM   STCGROUP    8192 Feb 21  2013 /var/empty
drwxr-xr-x   2 DB2ADM   STCGROUP    8192 Jan 29 15:09 /var/run
```

Check the permissions, extended attributes, and owner of the following files:

```
$ ls -El /usr/sbin/sshd
-rwxr--r--❶ ap--  2 DB2ADM❷  IPGROUP  6950912 Jan 16 14:30 /usr/sbin/sshd

$ ls -El /bin/ssh* /bin/scp /bin/sftp
-rwxr-xr-x  a-s-❸ 2 DB2ADM   IPGROUP  5210112 Apr 15  2013 /bin/scp
-rwxr-xr-x  a-s-  2 DB2ADM   IPGROUP  5292032 Apr 15  2013 /bin/sftp
-rwxr-xr-x  ----  2 DB2ADM   IPGROUP  6807552 Apr 15  2013 /bin/ssh
-rwxr-xr-x  --s-  2 DB2ADM   IPGROUP  5529600 Apr 15  2013 /bin/ssh-add
-rwxr-xr-x  --s-  2 DB2ADM   IPGROUP  5316608 Apr 15  2013 /bin/ssh-agent
-rwxr-xr-x  --s-  2 DB2ADM   IPGROUP  5632000 Apr 15  2013 /bin/ssh-keygen
-rwxr-xr-x  --s-  2 DB2ADM   IPGROUP  5615616 Apr 15  2013 /bin/ssh-keyscan

$ ls -El /usr/lib/ssh
drwxr-xr-x        2 DB2ADM   IPGROUP     8192 Oct 22  2011 IBM
-rwxr-xr-x  a-s-  2 DB2ADM   IPGROUP  1040384 Apr 15  2013 sftp-server
-rwxr-xr-x  --s-  2 DB2ADM   IPGROUP  3866624 Oct 22  2011 ssh-askpass
-rwsr-xr-x  ----  2 DB2ADM   IPGROUP  5967872 Dec 16 15:31 ssh-keysign
-rwxr-xr-x  --s-  2 DB2ADM   IPGROUP  4468736 Apr 15  2013 ssh-rand-helper
```

❶ The permissions bits should match this column.

❷ The owner must be UID=0; one of your UID=0 userids should be displayed.

❸ The extended attributes should match this column. `a`="APF authorized" `p`="Program Controlled" `s`="allow shared address space"

Reference: *PTUG:* "Steps for verifying the prerequisites for using OpenSSH"

1.5 Language Environment Tuning

IBM Ported Tools z/OS uses the LE XPLINK libraries, and IBM recommends the following:

- Add SCEELPA to LPALST
- Add SCEERUN and SCEERUN2 to LNKLST
- SCEERUN and SCEERUN2 must be program controlled
- Implement samples SCEESAMP(CEEWLPA) and SCEESAMP(EDCWLPA). We recommend implementing both of these as shipped.

Note: IBM Ported Tools OpenSSH will still run if recommended XPLINK modules are not placed in LPA. This is something that you can defer for your next system maintenance window.

References:
- *PTUG:* "Setting up the XPLINK environment for use by IBM Ported Tools for z/OS: OpenSSH" (Chapter 4)
- *Language Environment Customization* "Placing Language Environment modules in link pack and LIBPACK"

1.6 Using ICSF and /dev/random

Generation of secure random numbers is key to using OpenSSH (or any cryptographic tool). IBM Ported Tools OpenSSH will automatically use the z/OS Unix `/dev/random` device if ICSF's `CSFRNG` service is available. In the past this required that you have a co-processor card, but with the "A0" or later level of ICSF (HCR77A0/A1) you don't need a co-processor card - ICSF will generate a cache of secure random numbers using CPACF instructions as appropriate.

Note: If you are running z/OS 1.10 or 1.11 or don't have ICSF HCR77A0 or later installed, then `/dev/random` will not be supported unless you have a co-processor card installed with ICSF. Without `/dev/random` support, OpenSSH will internally use the **ssh-rand-helper** command, which can add several seconds to the startup of each session. It can also cause timeouts and generally requires that the ssh client and sshd server (SSHDAEM) have write access to their home directories. Search the PTUG for **"ssh-rand-helper"** for other setup issues.

Assuming that ICSF is running and supports the `CSFRNG` service, all you need to do is to authorize your users to this service. For most environments, it will be acceptable to permit all users to the CSFRNG service:

```
RDEFINE CSFSERV CSFRNG UACC(NONE)
PERMIT CSFRNG CLASS(CSFSERV) ID(*) ACCESS(READ)
SETROPTS RACLIST(CSFSERV) REFRESH
```

To verify that `/dev/random` is working, issue this command from a z/OS UNIX shell and userid with normal priviledges (and CSFRNG access). This should display some random data in hex:

```
$ head /dev/random | od -x
```

Reference: *PTUG:* "Using hardware support to generate random numbers"

1.7 Creating configuration files

Copy the sample configuration files to the `/etc/ssh` directory. You must use a UID=0 userid for this:

```
$ cd /samples
$ cp -p moduli          /etc/ssh
$ cp -p ssh_config      /etc/ssh
$ cp -p ssh_prng_cmds   /etc/ssh
$ cp -p sshd_config     /etc/ssh
$ cp -p zos_ssh_config  /etc/ssh
$ cp -p zos_sshd_config /etc/ssh
```

Note: All of the above files in `/etc/ssh` should be owned by a UID=0 userid and have permissions `644`:

```
-rw-r--r--   1 DB2ADM   STCGROUP  126379 May 17  2013 moduli
-rw-r--r--   1 DB2ADM   IPGROUP     2463 Nov 18 07:27 ssh_config
-rw-r--r--   1 DB2ADM   IPGROUP     5685 Oct 22  2011 ssh_prng_cmds
-rw-r--r--   1 DB2ADM   IPGROUP     4244 Jun 24  2013 sshd_config
-rw-r--r--   1 DB2ADM   IPGROUP     1231 Mar  4  2013 zos_ssh_config
-rw-r--r--   1 DB2ADM   IPGROUP     1282 Mar  4  2013 zos_sshd_config
```

Reference: *PTUG:* "Steps for creating or editing configuration files"

1.8 Creating SSHD server keys

You must generate one or more public/private key pairs that are used for authentication of your SSHD server. Each client that connects to the server will either already have one of the public keys (aka "host fingerprint") or will be required to accept your server's public key as proof of the server's identity.

For more information on SSH key authentication, see the recording of our webinar: *IBM Ported Tools for z/OS: Key Authentication*

Server keys can be stored either in protected UNIX files or in SAF/RACF keyrings. Most installations will choose to use files, which is covered below. For information on how to use SAF/RACF keyrings, see our webinar: *IBM Ported Tools for z/OS: Using Key Rings*

The following commands can be executed by a UID=0 userid to create DSA and RSA server key pairs in UNIX files:

```
$ cd /etc/ssh
$ ssh-keygen -t dsa -f ssh_host_dsa_key -N ""
Generating public/private dsa key pair.
Your identification has been saved in ssh_host_dsa_key.
Your public key has been saved in ssh_host_dsa_key.pub.
The key fingerprint is:
7e:a3:fb:db:c6:9f:16:d7:96:6c:ae:1d:bb:33:20:39 KIRK@ZOS1

$ ssh-keygen -t rsa -f ssh_host_rsa_key -N ""
Generating public/private rsa key pair.
Your identification has been saved in ssh_host_rsa_key.
Your public key has been saved in ssh_host_rsa_key.pub.
The key fingerprint is:
22:cb:9c:30:9d:98:c8:4f:45:a8:ac:00:e5:8e:62:af KIRK@ZOS1
```

This should result in two pairs of private/public key files with the following permissions, all owned by a UID=0 userid:

```
$ ls -al *key*
-rw-------   1 DB2ADM   OMVSGRP      668 Feb  4  2014 ssh_host_dsa_key
-rw-r--r--   1 DB2ADM   OMVSGRP      601 Feb  4  2014 ssh_host_dsa_key.pub
-rw-------   1 DB2ADM   OMVSGRP     1675 Feb  4  2014 ssh_host_rsa_key
-rw-r--r--   1 DB2ADM   OMVSGRP      393 Feb  4  2014 ssh_host_rsa_key.pub
```

Reference: *PTUG:* "Steps for setting up server authentication when keys are stored in UNIX files"

1.9 Set up SSHD server userids

Your SSHD server will use two SAF/RACF userids (besides the actual userids that clients sign on with):

1. The priviledged UID=0 userid used to start the started task (here we use "SSHDAEM")

2. The unpriviledged "privilege separation" userid (this must be "SSHD", or have an alias of "SSHD")

The priviledged started task userid can be an existing UID=0 userid, like OMVSKERN, but we recommend creating a new userid, defined like OMVSKERN:

```
ADDUSER SSHDAEM DFLTGRP(OMVSGRP)
  OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
  NOPASSWORD
```

The privileged started task userid must have read access to BPX.DAEMON -

```
RDEFINE FACILITY BPX.DAEMON UACC(NONE)
PERMIT BPX.DAEMON CLASS(FACILITY) ID(SSHDAEM) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Note: If your system has the FACILITY/BPX.POE defined, then the SSHDAEM userid will require READ access. If the SERVAUTH class is active, then the SSHDAEM userid will require authorization to access incoming socket connections.

Create the unpriviliged "privilege separation" userid, where `ggg` is an unused groupid and `uuu` is an unused non-zero uid (you may alternatively use the `AUTOGID` and `AUTOUID` keywords if you have enabled the BPX.NEXT.USER profile) -

```
ADDGROUP SSHDG OMVS(GID(ggg))
ADDUSER SSHD DFLTGRP(SSHDG) OMVS(UID(uuu) HOME('/var/empty')
  PROGRAM('/bin/false')) NOPASSWORD
```

Reference: *PTUG:* "Step for creating the sshd privilege separation user" and "Starting sshd as a stand-alone daemon"

1.10 Create SSHD server started task

The best way to start your SSHD server is by using a started task proc:

```
//SSHD PROC
//SSHD EXEC PGM=BPXBATCH,REGION=0M,TIME=NOLIMIT,
//     PARM='PGM /bin/sh -c /etc/ssh/sshd.sh'
//STDERR DD SYSOUT=*
//
```

This proc executes a shell script `/etc/ssh/sshd.sh` that you must create:

```
#!/bin/sh
export NLSPATH=$NLSPATH:/usr/lib/nls/msg/%L/%N.cat
export _EDC_ADD_ERRNO2=1
nohup /usr/sbin/sshd -f /etc/ssh/sshd_config &
sleep 1
```

This script should have permissions "700" and be owned by a UID=0 userid.

```
-rwx------   1 DB2ADM   STCGROUP     141 Feb 26  2013 sshd.sh
```

The SSHD started task must be configured to start with the privileged userid "SSHDAEM" that you setup in the prior section:

```
SETROPTS GENERIC(STARTED)
RDEFINE STARTED SSHD.*
  STDATA(USER(SSHDAEM) GROUP(OMVSGRP) TRUSTED(NO))
SETROPTS RACLIST(STARTED) REFRESH
```

To start sshd, issue the following MVS command:

```
S SSHD
```

Verify that the proper userid and group were assigned to the SSHD started task by examining the system log:

```
S SSHD
$HASP100 SSHD     ON STCINRDR
IEF695I START SSHD     WITH JOBNAME SSHD     IS ASSIGNED TO USER
SSHDAEM, GROUP OMVSGRP
$HASP373 SSHD     STARTED
$HASP395 SSHD     ENDED
```

Note: like FTPD, this started task will quickly terminate, but it will spin off an OMVS address space with jobname "SSHDn".

References:

- *PTUG:* "Starting sshd as a stand-alone daemon"
- *PTUG:* "Ways to start sshd as a stand-alone daemon" / "Using BPXBATCH"

1.11 TCP configuration

Using the default `sshd_config` settings, SSHD listens on port 22 on all stacks. Since this is a privileged port number, only programs running as superuser are allowed to listen on this port. We recommend that for your next IPL that you also reserve this port in `PROFILE.TCPIP`, since it also serves to document usage. In the following example template, we also cause the SSHD started task to be started after TCPIP has started:

```
AUTOLOG
...
    SSHD                ; SSHD Server (STC SSHDn)
...
ENDAUTOLOG
...
PORT
...
    22 TCP SSHD* NOAUTOLOG  ; Ported Tools SSHD server
...
```

Reference: *PTUG:* "Steps for creating or editing configuration files" Step 4

1.12 Verify z/OS DNS / Resolver operation

The ssh client will perform DNS lookups on target host names. Also, by default, the sshd server will lookup remote host names and check that the resolved name maps back to the IP address (warnings will be logged otherwise) . If the z/OS DNS client (the "Resolver") is not working properly, these requests might hang before timing out.

A full z/OS DNS server is not required to run IBM Ported Tools OpenSSH, but most shops will want to run a `"caching-only server"` connected to their corporate DNS servers. At minimum you should at least configure the z/OS resolver so that DNS requests do not hang.

To verify that your z/OS resolver is working properly, issue the following command from a z/OS UNIX shell. Try known and unknown host names to verify that neither hang:

```
$ host www.ibm.com
EZZ8321I e3062.x.akamaiedge.net has addresses 23.67.232.41
EZZ8322I aliases: www.ibm.com, www.ibm.com.cs186.net, www.ibm.com.edgekey.net

$ host h42444.not-a-domain.com
EZZ8342I h12345.not-a-domain.com: Unknown host
```

References:

- *PTUG:* "Troubleshooting" / "DNS is not configured properly"
- *z/OS Comm Svr IP Confg Guide* "The resolver" and "Domain Name System"

1.13 Configuring the syslogd daemon

The IBM Ported Tools SSHD server will log messages to the z/OS Communications Server **syslogd** facility using a local UNIX datagram socket (`/dev/log`).

IBM FTPD, TELNETD, IDS, and other z/OS Comm Server applications use **syslogd** for logging and tracing as well. Although some of these z/OS Comm Server applications will log to the z/OS console if **syslogd** is not running, IBM Ported Tools SSHD server will not. Therefore, it is important to have **syslogd** running in local mode (**-i**) so that SSHD server log messages are available.

Starting with z/OS 1.11, significant enhancements were made to **syslogd** -

- performance improvements
- improved operator interfaces
- automatic archival to MVS data sets
- ISPF syslogd viewer

IBM Ported Tools SSHD server by default will log all messages with `INFO` severity or higher to the local syslogd `"AUTH"` facility.

You should verify that **syslogd** is configured and started on your z/OS system and that your `/etc/syslog.conf` file is configured so that messages to the `AUTH` facility will be logged to some file. For example:

```
# log any messages to AUTH facility (sshd)
auth.* /tmp/syslogd.auth.log -X
```

References:

- *z/OS Comm Svr IP Confg Guide* "Configuring the syslog daemon"
- Presentation: *z/OS V1R11 Comm Server - syslogd enhancements*

1.14 Verify basic functionality

Before we move on, lets verify the basic functionality of your SSHD server. To do this, you will need to install an SSH client, like *PuTTY* on your workstation on a network that can connect to your z/OS system on port 22.

Note: You will want to test an ssh session to an unprivileged z/OS userid that has an OMVS segment and home directory. If you have a non UID=0, unprivileged TSO userid that can get into the TSO OMVS shell, then use that.

```
C:> putty zosuser@my.zos.host
 - or -
C:> putty zosuser@10.2.3.4
Using username "zosuser".
zosuser@my.zos.host's password: ******
/u/home/zosuser>
```

If successful, the above will place you in an SSH session with an interactive z/OS UNIX login shell. Once you have a non-TSO UNIX shell, you can use the ssh client command to connect to other hosts. For example, you can also connect to the same z/OS system using the loopback address:

```
/u/home/zosuser> ssh zosuser@127.0.0.1
The authenticity of host '127.0.0.1 (127.0.0.1)' can't be established.
RSA key fingerprint is 22:cb:9c:30:9d:98:c8:4f:45:a8:ac:00:e5:8e:62:af.
Are you sure you want to continue connecting (yes/no)? yes
zosuser@my.zos.host's password: ******
/u/home/zosuser>
/u/home/zosuser> exit
/u/home/zosuser> (back to the first connection)
```

Note: The IBM Ported Tools ssh client will not run under a TSO OMVS shell. You can only use it from another ssh shell, a (non-3270) telnet shell, or a batch job.

Now is also a good time to verify your **syslogd** configuration. Look at the end of the logfile that you configured in `/etc/syslog.conf`:

```
$ tail /tmp/syslogd.auth.log
...
Feb 14 21:11:23 S0W1 sshd[67174502]: Port of Entry information retained for ...
Feb 14 21:11:24 S0W1 sshd[67174502]: Accepted publickey for kirk from ...
```

2. Exploiting ICSF acceleration

2.1 Enabling ICSF Cipher and MAC support

To use ICSF to accelerate Cipher and MAC algorithms in IBM Ported Tools OpenSSH, all z/OS userids that use the ssh client or login to the sshd server, including "SSHDAEM" and "SSHD" (the SSHD privileged and privilege separation userids) will require read access to the following SAF/RACF profiles in the **CSFSERV** class:

- CSFIQA - ICSF Query Algorithm
- CSF1TRC - PKCS #11 Token record create
- CSF1TRD - PKCS #11 Token record delete
- CSF1SKE - PKCS #11 Secret key encrypt
- CSF1SKD - PKCS #11 Secret key decrypt
- CSFOWH - One-Way Hash Generate

Some installations may wish to create a SAF/RACF `GROUP` with access to these profiles and then connect required users to this group. Given the nature of these services, many installations will choose to permit all users to these resources:

```
RDEFINE CSFSERV CSFIQA UACC(NONE)
RDEFINE CSFSERV CSF1TRC UACC(NONE)
RDEFINE CSFSERV CSF1TRD UACC(NONE)
RDEFINE CSFSERV CSF1SKE UACC(NONE)
RDEFINE CSFSERV CSF1SKD UACC(NONE)
RDEFINE CSFSERV CSFOWH  UACC(NONE)

PERMIT  CSFIQA  CLASS(CSFSERV) ID(*) ACCESS(READ)
PERMIT  CSF1TRC CLASS(CSFSERV) ID(*) ACCESS(READ)
PERMIT  CSF1TRD CLASS(CSFSERV) ID(*) ACCESS(READ)
PERMIT  CSF1SKE CLASS(CSFSERV) ID(*) ACCESS(READ)
PERMIT  CSF1SKD CLASS(CSFSERV) ID(*) ACCESS(READ)
PERMIT  CSFOWH  CLASS(CSFSERV) ID(*) ACCESS(READ)

SETROPTS CLASSACT(CSFSERV)
SETROPTS RACLIST(CSFSERV) REFRESH
```

Note: Starting with ICSF version HCR77A0, the **CSF1TRC** API used by P.T. OpenSSH will check for the existence of a SAF/RACF resource: `CLEARKEY.SYSTOK-SESSION-ONLY CLASS(CRYPTOZ)`. IBM suggests that you do you not define this specific resource or a matching generic resource. If you do, then you must permit all SSH/SSHD userids `READ` access.

If you are using ICSF version HCR77A1, then you should definitely consider defining the following two profiles. This will disable SAF/RACF checking for **CSFRNG** (random number; `/dev/random`) and **CSFOWH** (MACs). Since MAC generation occurs for each SSH packet, this can save a couple of points off CPU usage.

```
RDEFINE XFACILIT CSF.CSFSERV.AUTH.CSFOWH.DISABLE
    UACC(READ)
RDEFINE XFACILIT CSF.CSFSERV.AUTH.CSFRNG.DISABLE
    UACC(READ)
SETROPTS CLASSACT(XFACILIT)
SETROPTS RACLIST(XFACILIT) REFRESH
```

Next, you must update *both* `/etc/ssh/zos_ssh_config` and `/etc/ssh/zos_sshd_config` and add the following two lines:

```
CiphersSource any
MACsSource any
```

The CiphersSource and MACsSource keywords are only supported if you have installed the PTF for APAR OA37278, so now is a good time to check this by verifying your sshd configuration files. From a UID=0 userid, enter the following command in the z/OS UNIX shell:

```
$ /usr/sbin/sshd -t
$ (should complete with no error messages)
```

Now, verify that you can still open an ssh connection from your workstation into the SSHD server. Then, using that z/OS UNIX shell session issue a dummy ssh client command to verify ICSF functionality. It should look something like this:

```
$ ssh -vv foo@bar
OpenSSH_5.0p1, OpenSSL 1.0.1c 10 May 2012
...
debug1: zsshVerifyIcsfSetup: ICSF FMID is 'HCR77A1 '
debug1: zsshVerifyIcsfSetup (163): CSFIQA successful: return code = 0, reason code = 0
debug2: ----------------------------------
debug2: CRYPTO   SIZE     KEY      SOURCE
debug2: ----------------------------------
debug2: AES      256      SECURE   COP
debug2: AES      256      SECURE   CPU
debug2: DES      56       SECURE   COP
debug2: DES      56       SECURE   CPU
...
debug2: SHA-1    160      NA       CPU
debug2: SHA-2    512      NA       CPU
debug2: TDES     168      SECURE   COP
debug2: TDES     168      SECURE   CPU
debug2: ssh_connect: needpriv 0
FOTS1336 ssh: Could not resolve hostname bar: EDC9501I The name does not resolve ...
```

In the above ICSF CRYPTO table display, we can see that:

- the AES Cipher is supported up through 256 bits. This corresponds to OpenSSH Cipher algorithms:

`aes128-cbc`, `aes192-cbc`, and `aes256-cbc`. Older processors support less than 256 bit AES Ciphers. The value listed here is the maximum strength support by your processor.

- SHA-1 is supported, which corresponds to OpenSSH MAC algorithms: `hmac-sha1` and `hmac-sha1-96`
- Triple-DES is supported, which corresonds to OpenSSH Cipher `3des-cbc`

Note: The entries with `SOURCE=CPU` correspond to ICSF facilities that are implemented with CPACF, which is what is used by IBM Ported Tools OpenSSH. For Ciphers and MACs, these are more efficient than using a co-processor anyway.

Reference: *PTUG:* "Setting up OpenSSH to use ICSF ciphers and MAC algorithms"

2.2 Configure OpenSSH Ciphers and MACs

In this section, you will configure the Cipher and MAC algorithms that your ssh client and sshd server will use. Choosing ICSF algorithms implemented via CPACF can result in CPU savings upward of 50%, so this is important to understand and implement correctly.

The default Cipher and MAC algorithm names can be seen (commented out) in the sample `/etc/ssh/ssh_config` (ssh client) and `/etc/ssh/sshd_config` (sshd server) configuration files. *Both* files have these two lines:

```
#  Ciphers aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,aes128-cbc,3des-cbc,
blowfish-cbc,cast128-cbc,aes192-cbc,aes256-cbc,arcfour,rijndael-cbc@lysator.liu.se

#  MACs hmac-md5,hmac-sha1,umac-64@openssh.com,hmac-ripemd160,hmac-ripemd160@openssh.c
om,hmac-sha1-96,hmac-md5-96
```

These lists specify the Cipher and MAC algorithms that are supported by the ssh client and server. When an ssh client connects to an ssd server, they exchange these lists and *negotiate* which Cipher and MAC algorithm will be used for this session.

Cipher and MAC negotiation rule

The first algorithm in the client list that appears anywhere in the server list will be selected.

For MACs, we recommend that you support only the ICSF/CPACF algorithms: `hmac-sha1` and `hmac-sha1-96`. One of these should always be supported by an ssh client or sshd server.

Configuring SSH client Ciphers and MACs

Considering the ssh client configuration first, and mindful of the rule above, we can choose between the following two general strategies:

1. Only allow ICSF/CPACF supported Ciphers and fail otherwise
2. Prefer ICSF/CPACF supported Ciphers, but allow others if the server doesn't support these

Other considerations:

- sites should prefer (list first) the minimum strength `aes*-cbc` algorithm that meets their requirements, followed by stronger `aes*-cbc` Ciphers up through the maximum bit length supported by CPACF on their processor (see the table in the previous section).
- sites may choose not to support the older 3des-cbc algorithm, or other non-ICSF Ciphers that do not meet their security requirements.
- The **Host** and **Match** configuration keywords can be used in SSH client configuration files to conditionally assign different algorithm lists to specific hosts, userids, etc.

To implement your strategy, update the `/etc/ssh/ssh_config` file and add the `MACs` line and *one* of the following `Cipher` lines (a single line, shown wrapped below)

```
# /etc/ssh/ssh_config changes

# Only support ICSF/CPACF SHA-1 MACs:
MACs hmac-sha1,hmac-sha1-96

# *** PICK (1) or (2) ***
# 1) Only allow AES ICSF/CPACF Ciphers, except 3des-cbc
Ciphers aes128-cbc,aes192-cbc,aes256-cbc

# 2) Prefer AES ICSF/CPACF Ciphers, but fallback to others
Ciphers aes128-cbc,aes192-cbc,aes256-cbc,aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,
arcfour128,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,rijndael-cbc@lysator.liu.se
```

Configuring SSHD server Ciphers and MACs

The negotiation rule implies that your have fewer choices for selecting Ciphers and MACs in your SSHD server configuration. Generally, there are three strategies:

1. Only allow ICSF/CPACF supported Ciphers and fail otherwise
2. Allow any Cipher; try to ask your clients to prefer ICSF/CPACF supported Ciphers
3. Allow a subset of the supported Ciphers as required by your clients

To implement your strategy, update the `/etc/ssh/sshd_config` file and add the `MACs` line and optionally a `Cipher` lines if you will only allow ICSF/CPACF supported Ciphers:

```
# /etc/ssh/sshd_config changes

# Only support ICSF/CPACF SHA-1 MACs:
MACs hmac-sha1,hmac-sha1-96

# *** CHOOSE (1), (2), or (3) -
# 1) Only allow AES ICSF/CPACF Ciphers, except 3des-cbc.
Ciphers aes128-cbc,aes192-cbc,aes256-cbc

# 2) Allow any Cipher; just leave the default line commented:
#  Ciphers aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,aes128-cbc,3des-cbc,
blowfish-cbc,cast128-cbc,aes192-cbc,aes256-cbc,arcfour,rijndael-cbc@lysator.liu.se

# 3) Allow a subset of Ciphers; uncomment the default/master list and remove some
...
```

2.3 Verifying ICSF usage

To verify that ICSF is being used in the IBM Ported Tools ssh client, you will need to open an ssh client connection from z/OS to some host (or to the same z/OS system). The ssh client command will not execute under a TSO OMVS session, but you can test it by first logging into z/OS with an ssh session from your workstation.

To display the necessary debugging information, invoke the ssh command from a z/OS Unix shell:

```
zos$ ssh -vvv myuser@127.0.0.1
...
debug3: RNG is ready, skipping seeding ❶
...
debug1: mac_setup_by_id: hmac-sha1 from source ICSF ❷
debug2: mac_setup: found hmac-sha1
debug1: zsshIcsfMacInit (402): CSFPTRC successful: return code = 0, reason code = 0,
 handle = 'SYSTOK-SESSION-ONLY 00000000S'
...
debug1: cipher_init: aes128-cbc from source ICSF ❸
debug1: zsshIcsfCipherInit (930): CSFPTRC successful: return code = 0, reason code = 0,
 handle = 'SYSTOK-SESSION-ONLY 00000001S'
...
```

❶ This message confirms that `/dev/random` is being used (and not ssh-rand-helper)
❷ These messages confirm that ICSF is being used for the selected MAC: hmac-sha1
❸ These messages confirm that ICSF is being used for the selected Cipher: aes128-cbc

To confirm that ICSF is being used for an IBM Ported Tools SSHD server session, you must enable debugging for SSHD by making the following temporary change in `/etc/ssh/sshd_config` and restart SSHD.

Note: this will generate lots of trace information to syslogd, so you will not want to do this when there (many) other sessions starting in this period. Also, you can do an in-place restart of SSHD by sending the "HUP" signal to the top-level daemon, which does not disrupt existing SSHD sessions. See the P.T User's Guide for details.

```
# file /etc/ssh/sshd_config
...
#SyslogFacility AUTH
#LogLevel INFO
LogLevel DEBUG3
...
```

Appendix A. Managing the /tmp filesystem

The `/tmp` filesystem is important to the operation of many applications that use z/OS UNIX services. Some common uses include:

- z/OS UNIX shell scripts often create temporary, transient work files. One example is setting a variable with the output of a command. Although tiny and short lived, a temporary file is required. If temp space is unavailable, then the script can fail in unexpected ways. If you will be running batch jobs that use Co:Z SFTP, IBM ssh or IBM sftp clients, you will likely be using the shell and need temp files.
- If you will be using Co:Z SFTP server, each session causes SSHD to invoke a shell script which configures the session.
- Co:Z SFTP server by default creates a log file for each session in `/tmp`. These are important to keep for some period of time:
  - The current session log file can be accessed by the remote sftp client (e.g: **get /+error.log**) to get details of a problem.
  - If there is a failure, support personnel can review the session log file for diagnostic information.
  - Trace messages, if enabled, will go to the session log.

  In many cases, installations will choose to put Co:Z SFTP server session logs in a separate zFS or HFS filesystem.

A.1 Best practices

Installations should review the references below, but here are some general suggestions:

1. Schedule a nightly job that runs the z/OS UNIX **skulker** command to clean up old files. For example, the following job (run as a superuser) uses the IBM skulker script to remove files from tmp that have not been accessed for 5 days:

```
//SKULKER EXEC PGM=COZBATCH
//STDIN   DD   *
 cd /tmp || exit 8
 /samples/skulker -R -l /dev/fd2 . 5
//
```

2. Monitor your temp filesystem(s) for full threshold conditions using the **FSFULL** mount parameter. Use the threshold messages to alert your operations personnel.

   Note: FSFULL monitoring of TFS filesystems is not supported prior to z/OS 2.1

3. Document procedures, commands, and tools to be used by personnel in the event of a full condition. Some useful commands include:

```
# display filesystem status
$ df -kP /tmp
```

```
# display 10 largest files in /tmp
$ du -aktx /tmp | sort -nr | head -10

# display pids(users) that are using a file
#   - should do this before removing a file to verify no users
$ fuser -u file

# display process/job info for a pid
$ ps -o pid,ppid,user,jobname,xasid,stime,time,comm -g <pid>

# truncate an in-use log file
#   - removing would leave an unnamed file until all using processes complete
$ cat /dev/null > file

# kill processes using a file
$ fuser -ku file

# (z/OS 2.1) list hidden in use files in a filesystem
$ zlsof -d /tmp

# (pre-z/OS 2.1) list users of deleted files
#   from IBM Tools & Toys:
#   http://www-03.ibm.com/systems/z/os/zos/features/unix/bpxa1ty2.html
$ delinuse /tmp
```

References:

- *z/OS UNIX System Services Planning* "Managing the temporary file system (TFS)"
- *z/OS UNIX System Services Command Reference* "skulker - Remove old files from a directory"